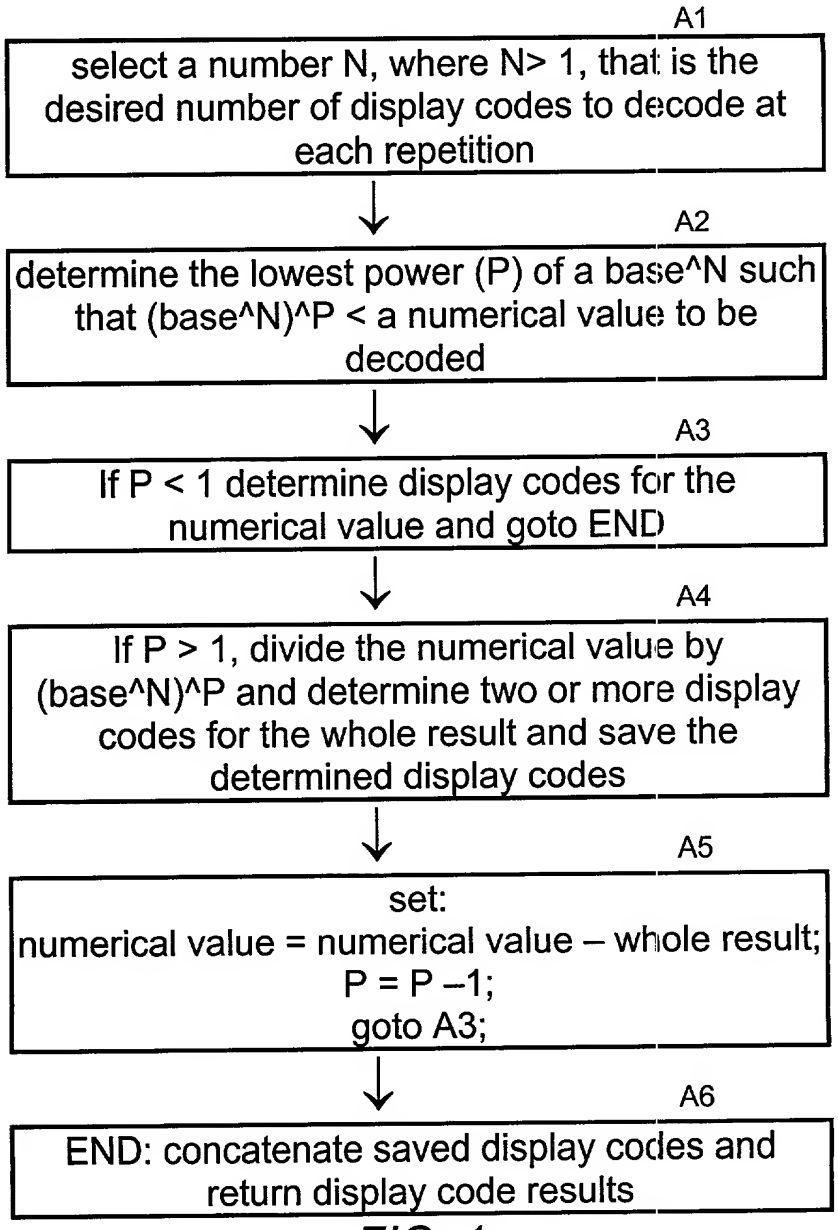

FIG. 1

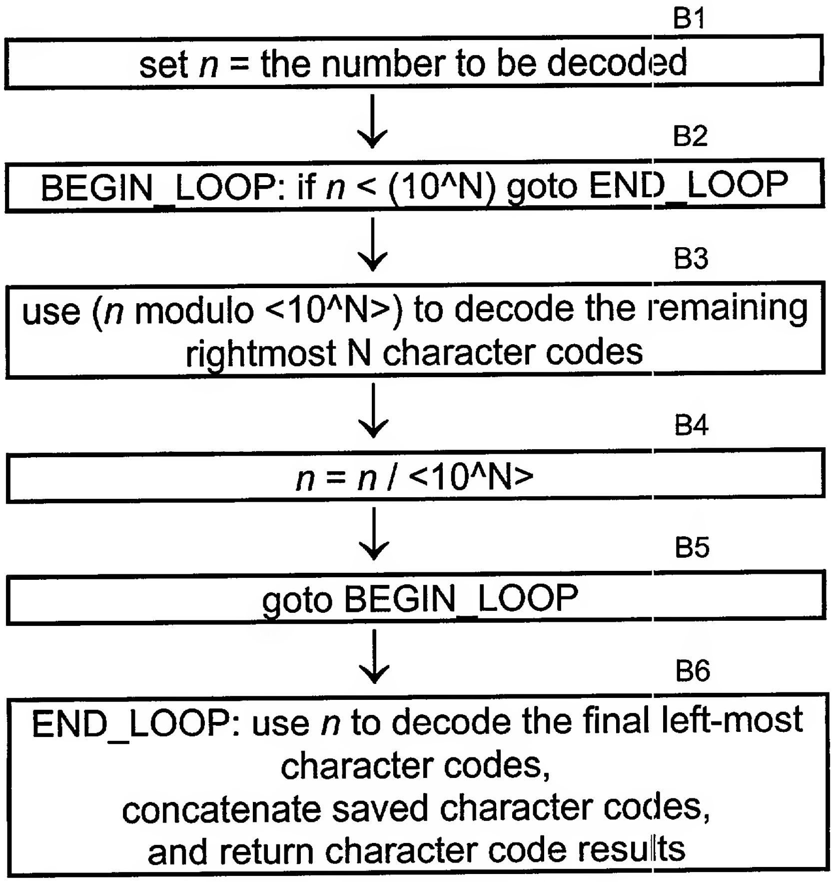
B1
set n = the number to be decoded
B2
BEGIN_LOOP: if n < (10^N) goto END_LOOP
B3
use (n modulo <10^N>) to decode the remaining rightmost N character codes
B4
n = n / <10^N>
B5
goto BEGIN_LOOP
B6
END_LOOP: use n to decode the final left-most character codes,
concatenate saved character codes,
and return character code results

FIG. 2

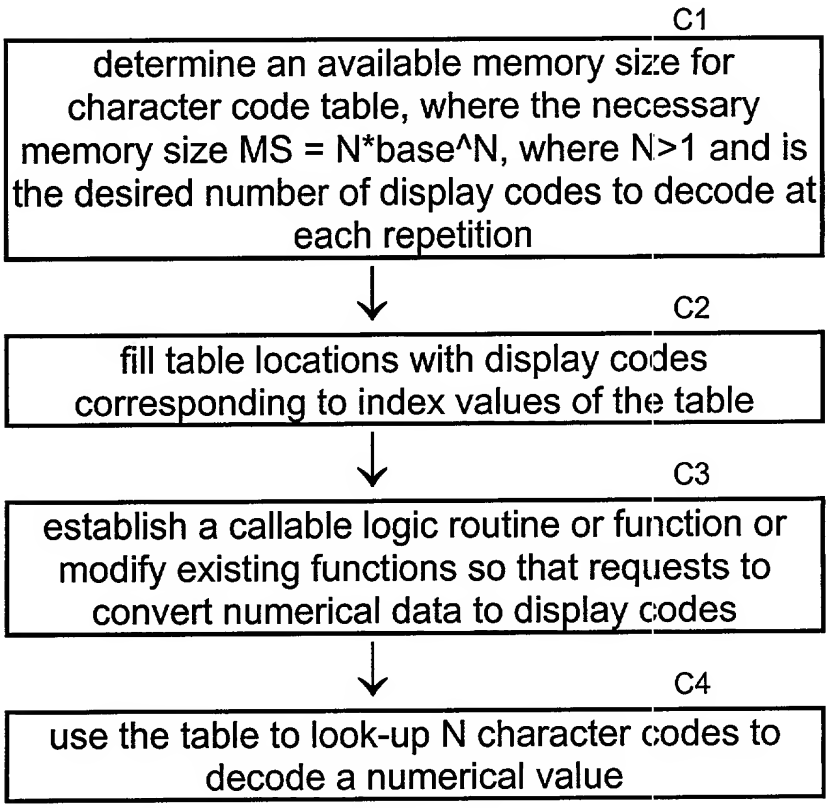

FIG. 3

FIG. 4 - ASCII Table (7-bit) for reference

Dec	Oct	Hex	Binary	Value
000	000	000	00000000	NUL(Null char.)
001	001	001	00000001	SOH(Start of Header)
002	002	002	00000010	STX(Start of Text)
003	003	003	00000011	ETX(End of Text)
004	004	004	00000100	EOT(End of Transm.)
005	005	005	00000101	ENQ(Enquiry)
006	006	006	00000110	ACK(Acknowledgment)
007	007	007	00000111	BEL(Bell)
008	010	008	00001000	BS(Backspace)
009	011	009	00001001	HT(Horizontal Tab)
010	012	00A	00001010	LF(Line Feed)
011	013	00B	00001011	VT(Vertical Tab)
012	014	00C	00001100	FF(Form Feed)
013	015	00D	00001101	CR(Carriage Return)
014	016	00E	00001110	SO
015	017	00F	00001111	SI
016	020	010	00010000	DLE(Data Link Escape)
017	021	011	00010001	DC1(XON)
018	022	012	00010010	DC2(Device Control)
019	023	013	00010011	DC3(XOFF)
020	024	014	00010100	DC4
021	025	015	00010101	NAK(Negative Ack)
022	026	016	00010110	SYN(Synchronous Idle)
023	027	017	00010111	ETB(End of T. Block)
024	030	018	00011000	CAN(Cancel)
025	031	019	00011001	EM
026	032	01A	00011010	SUB
027	033	01B	00011011	ESC(Escape)
028	034	01C	00011100	FS(File Separator)
029	035	01D	00011101	GS
030	036	01E	00011110	RS(Request to Send)
031	037	01F	00011111	US
032	040	020	00100000	SP(Space)
033	041	021	00100001	!
034	042	022	00100010	"
035	043	023	00100011	#
036	044	024	00100100	$
037	045	025	00100101	%
038	046	026	00100110	&
039	047	027	00100111	'
040	050	028	00101000	(
041	051	029	00101001	)
042	052	02A	00101010	*
043	053	02B	00101011	+
044	054	02C	00101100	,
045	055	02D	00101101	-
046	056	02E	00101110	.
047	057	02F	00101111	/
048	060	030	00110000	0
049	061	031	00110001	1
050	062	032	00110010	2
051	063	033	00110011	3
052	064	034	00110100	4
053	065	035	00110101	5
054	066	036	00110110	6
055	067	037	00110111	7
056	070	038	00111000	8
057	071	039	00111001	9
058	072	03A	00111010	:
059	073	03B	00111011	;
060	074	03C	00111100	<
061	075	03D	00111101	=
062	076	03E	00111110	>
063	077	03F	00111111	?

Dec	Oct	Hex	Binary	Value
064	100	040	01000000	@
065	101	041	01000001	A
066	102	042	01000010	B
067	103	043	01000011	C
068	104	044	01000100	D
069	105	045	01000101	E
070	106	046	01000110	F
071	107	047	01000111	G
072	110	048	01001000	H
073	111	049	01001001	I
074	112	04A	01001010	J
075	113	04B	01001011	K
076	114	04C	01001100	L
077	115	04D	01001101	M
078	116	04E	01001110	N
079	117	04F	01001111	O
080	120	050	01010000	P
081	121	051	01010001	Q
082	122	052	01010010	R
083	123	053	01010011	S
084	124	054	01010100	T
085	125	055	01010101	U
086	126	056	01010110	V
087	127	057	01011111	W
088	130	058	01011000	X
089	131	059	01011001	Y
090	132	05A	01011010	Z
091	133	05B	01011011	[
092	134	05C	01011100	\
093	135	05D	01011101	]
094	136	05E	01011110	^
095	137	05F	01011111	_
096	140	060	01100000	`
097	141	061	01100001	a
098	142	062	01100010	b
099	143	063	01100011	c
100	144	064	01100100	d
101	145	065	01100101	e
102	146	066	01100110	f
103	147	067	01100111	g
104	150	068	01101000	h
105	151	069	01101001	i
106	152	06A	01101010	j
107	153	06B	01101011	k
108	154	06C	01101100	l
109	155	06D	01101101	m
110	156	06E	01101110	n
111	157	06F	01101111	o
112	160	070	01110000	p
113	161	071	01110001	q
114	162	072	01110010	r
115	163	073	01110011	s
116	164	074	01110100	t
117	165	075	01110101	u
118	166	076	01110110	v
119	167	077	01110111	w
120	170	078	01111000	x
121	171	079	01111001	y
122	172	07A	01111010	z
123	173	07B	01111011	{
124	174	07C	01111100	|
125	175	07D	01111101	}
126	176	07E	01111110	~
127	177	07F	01111111	DEL

Hex Digits 1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO610000	Ø LO620000	° SM190000	µ SM170000	^ SD150000	{ SM110000	} SM140000	\\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	• SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	¢ SC040000	! SP020000	¦ SM650000	: SP130000	« SP170000	ª SM210000	¡ SP030000	[SM060000	-(SHY) SP320000	¹ SND011000	² ND021000	³ ND031000
-B	. SP110000	$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000	] SM080000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Ð LD620000	¯ SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000	) SP070000	_ SP090000	' SP050000	ý LY110000	¸ SD410000	Ý LY120000	¨ SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	´ SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	| SM130000	h SM660000	? SP150000	" SP040000	± SA020000	¤ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	(EO)

FIG. 5

```
//////////////////////////////////////////////////////////////////
// GET_ASCII_FRACTIONAL_DIGITS EXAMPLE CODE
//////////////////////////////////////////////////////////////////
// Brief description:    Get ascii fractional digits.
// Expected:
//            a       =   address of a character array for output ascii digits
//            i       =   index into above character array
//            n       =   binary integer to convert
//            p       =   scale fractional digits
//            digits  =   number of precision digits
// Result   :  i new index into a
// Other variables:
//             TABLE_SCALE = number of entries in looked-up table; also 10^N
//             index = index value used to access ntab
//             ntab = indexed table of character codes
//             TABLE_DIGITS = number of digits in each table entry
//             table_digit = position of digit (1st (0) through TABLE_DIGITS^th
//             TABLE_DIGITS -1) ) read in a
//             particular table entry; DAN, I AM A LITTLE CONFUSED BY THIS
//////////////////////////////////////////////////////////////////
inline int routine get_ascii_fractional_digits (char *a, int i, int n,
                          int fract_digits, int digits)
{
  int table_digit, index;
  if (n < 0)                              /* calculate -1 */
    n = -n;                               /* as 1 */
  while (fract_digits > 0)
    {
      if (n < TABLE_SCALE)
      {                                   /* already less no need for a divide */
        index = (int) n;                  /* use the number as the index */
        n = 0;                            /* no more number */
      }
      else
      {
        index = n % TABLE_SCALE;          /* set index to remainder (using modulo)  */
        n /= TABLE_SCALE;                 /* new n = n/TABLE_SCALE
      }
      for (table_digit = 0; table_digit < TABLE_DIGITS;
         table_digit++, --fract_digits)
        if (fract_digits > 0 && fract_digits <= digits)
          a[i++] = ntab[index][table_digit]; /* In this example, each character is
                                                /* separately fetched from its table
                                             /* entry and the table is treated as a
                                             /* 2-dimensional table.
    }
  return i;
}                                         /* END get ascii fractional digits */
```

FIG. 6

```
////////////////////////////////////////////////////////////////////
// GET_ASCII_WHOLE_DIGITS
////////////////////////////////////////////////////////////////////
// Brief description:  Get ascii whole integer digits.
// Expected:
//            a       =   address a character array for ascii digits
//            i       =   index into above character array
//            n       =   binary 64-bit integer to convert
//
// Result  :  i new index into a
//
////////////////////////////////////////////////////////////////////
inline int routine get_ascii_whole_digits (char *a, int i, real * _n)
{
  int index, table_digit, skip_leading;
  int64 n = _n->ip;                     /* integer portion */
  do
    {
      if (n < TABLE_SCALE)
      {                                 /* already less, no need for a divide */
        index = (int) n;                /* use the number as the index */
        n = 0;                          /* no more number */
      }
      else
      {
        index = (int) (n % TABLE_SCALE);  /* remainder */
        n /= TABLE_SCALE;                 /* divide by TABLE_SCALE */
      }

      a[i++] = ntab[index][0];          /* always at least 1 character */
      if (n)                            /* n does not equal zero */
      {                                 /* more to come */
        for (table_digit = 1; table_digit < TABLE_DIGITS;
            table_digit++)
          a[i++] = ntab[index][table_digit];/* convert all digits for this entry*/
      }
      else
      {                                   /* skip leading '0's on last divide */
        for (skip_leading = TABLE_DIGITS - 1; skip_leading;
            --skip_leading)
          if (ntab[index][skip_leading] != '0')
            {
              for (table_digit = 1; table_digit <= skip_leading;
                  table_digit++) a[i++] = ntab[index][table_digit];
              break;
            }
      }
    }
  while (n);
  return i;
}                                     /* get ascii whole digits */
```

FIG. 7

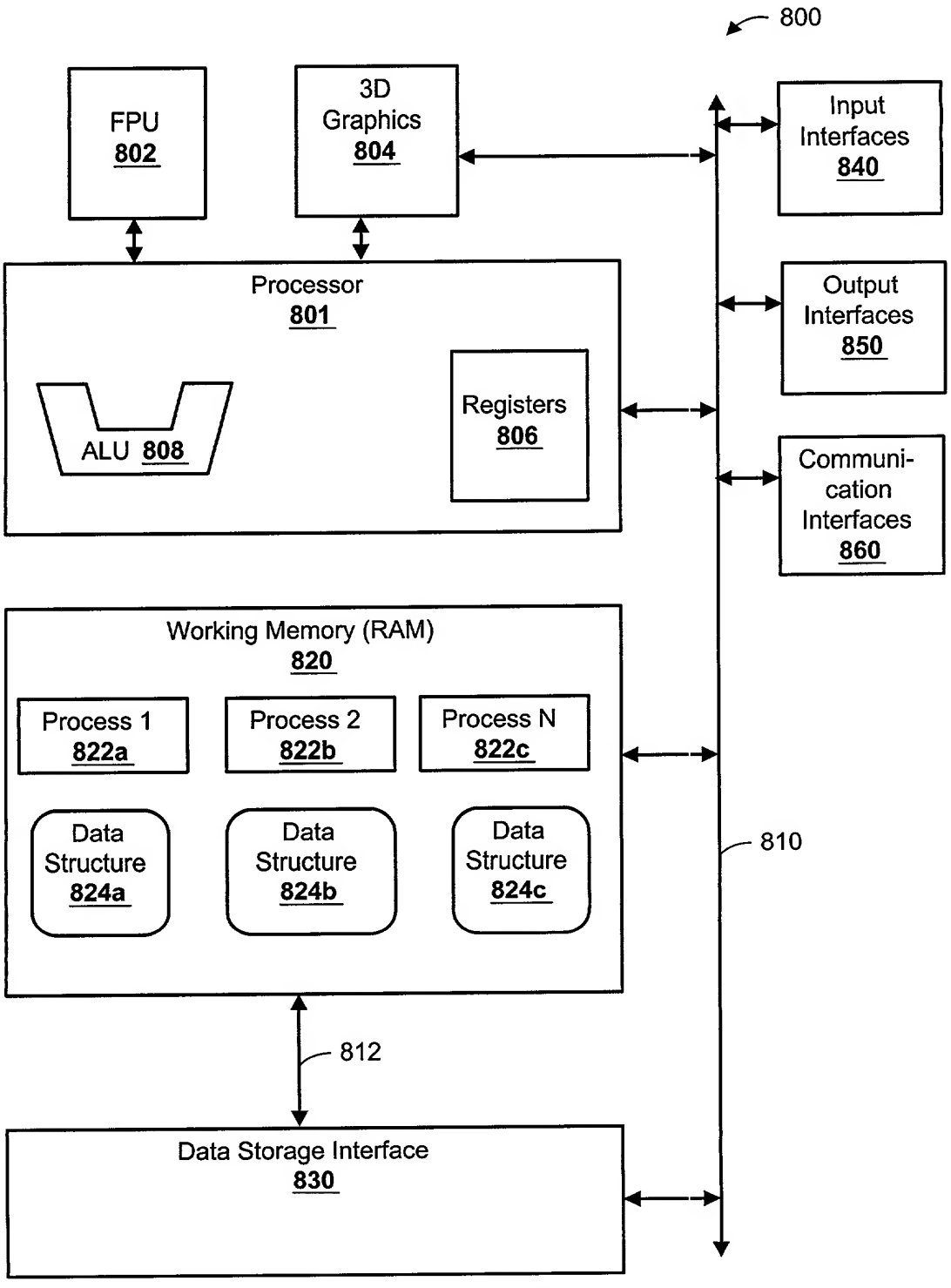
800
FPU
802
3D
Graphics
804
Input
Interfaces
840
Processor
801
Output
Interfaces
850
ALU 808
Registers
806
Communi-
cation
Interfaces
860
Working Memory (RAM)
820
Process 1
822a
Process 2
822b
Process N
822c
Data
Structure
824a
Data
Structure
824b
Data
Structure
824c
810
812
Data Storage Interface
830

FIG. 8

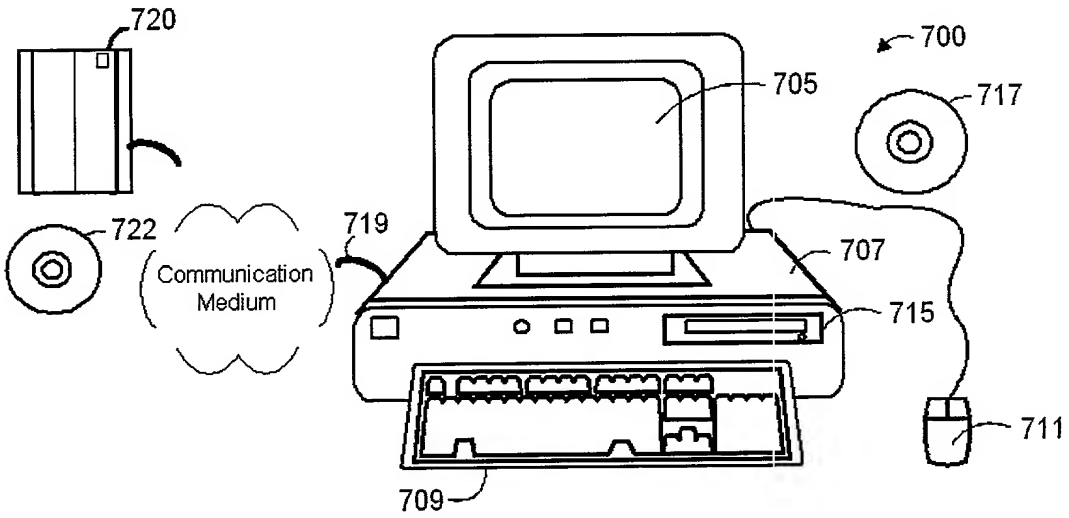

FIG. 9